auxfirst agency

AGENT OPERABILITY

THE AGENT-OPERABLE ENTERPRISE

Why your workflows - not your models - decide whether agents work

01	02	03
DATA SHAPE	PROCESS DESIGN	TRUST + PERMISSIONS

A field guide for enterprise AI and technology leaders

First edition · July 2026

A note from auxfirst

For forty years, enterprise software assumed that a human would be present at the decisive moment: reading the screen, interpreting the exception, choosing the next step, and carrying the accountability. AI agents change that assumption.

The moment software can remember context, choose among options, call tools, and initiate action, the design problem is no longer limited to the interface. It becomes a question of delegation: what work can move, what authority can travel with it, what evidence must be left behind, and how a human recovers control.

auxfirst calls the wider discipline **Agentic Experience Design (AUX)**. AUX designs the enduring relationship between people and systems that remember, adapt, and act. Trust is not a feature added at the end. It is the operating condition that makes delegation possible.

This field guide applies that point of view to the enterprise itself. It introduces **agent operability**: a practical way to assess whether a workflow can be safely and repeatably operated by an agent, rather than merely observed or summarized by one.

The thesis is simple:

> **A capable agent inside an inoperable workflow is still an inoperable system.**

CONTENTS

Introduction — The pilot that died in security review A capable agent, a promising pilot, and the unanswered permissions question that stopped everything.

1 — What IT leaders are actually saying Why change management, data readiness, and workflow redesign have become the real enterprise AI agenda.

2 — The three layers of agent operability Data shape, process design, and trust & permissions — the load-bearing model for the book.

3 — Data shape: readable is not operable The maturity ladder from information a human can find to context an agent can safely act on.

4 — Process design: the Action Heat Ladder How to redraw work across read, draft, decide, and execute — and set the right control posture for every action.

5 — The permissions problem nobody owns Why agents need their own identity, permission envelope, sponsor, and accountability trail.

6 — Trust architecture: making agent output answerable Authorship, evidence, auditability, escalation, and continuous conformance.

7 — The new operating roles The internal FDE pattern and the cross-functional ownership model required to make agents operational.

8 — What headless means for your vendor stack How to test whether enterprise software can be used by agents technically, securely, and commercially.

9 — The 90-day path to your first agent-operable workflow A practical sequence from workflow selection to an evidence-backed build, buy, or wait decision.

Conclusion — Operability compounds

Appendix A — Agent Operability Self-Assessment

Appendix B — Glossary: the vocabulary of agent operability

Selected references

INTRODUCTION

The pilot that died in security review

The pilot looked excellent in the demonstration.

A procurement agent could read an incoming vendor questionnaire, locate prior answers, pull the right policy language, identify gaps, and produce a response pack in minutes. The model was capable. The retrieval worked. The output was cleaner than the team expected. The sponsor could see weeks of effort disappearing from the process.

Then security asked a basic question.

What account will the agent use?

The room went quiet.

The workflow crossed five systems. The procurement analyst could access three of them. Legal could access two. Information security could see the policy repository but not the commercial records. A senior manager could approve exceptions, but only inside a separate ticketing flow. No individual human had the combined access the agent would need to execute the workflow end to end.

Someone suggested using the procurement lead's login. Someone else suggested a shared service account. The pilot team said the agent could simply ask a human whenever it reached a protected step. Security asked how the handoff would be logged, how privileges would expire, how data would be isolated between cases, who would own the output, and whether the agent could distinguish a routine exception from a legally material one.

There were no answers — only implementation ideas.

The pilot did not fail because the model hallucinated. It did not fail because the prompt was weak. It failed because the enterprise had treated the workflow as a sequence of screens rather than a system of data, decisions, authority, and accountability.

That is the pattern now repeating across organizations. Agents perform well in a bounded demo because the demo quietly removes the difficult parts. The data is preselected. The exceptions are controlled. Permissions are borrowed from the person running the test. The outputs remain drafts. The cost of being wrong is low because nothing has actually moved.

Production puts the difficulty back in.

The agent meets inconsistent records, missing fields, old policies, ambiguous ownership, undocumented judgment, conflicting incentives, and tools that were designed for a person with a browser. It needs access that no single role currently holds. It encounters an exception whose meaning exists only in the memory of one experienced employee. It creates an output that is technically generated by software but commercially owned by the company. Suddenly, the model is the least interesting part of the system.

The operating-model problem wearing a technology costume

Enterprise AI discussions often begin with capability: which model reasons best, which platform orchestrates agents, which vendor has the strongest roadmap. Those choices matter. But capability is not operability.

A forklift can lift more than a person. That does not make every building forklift-operable. The aisles need clearance. The floor needs load capacity. The goods need pallets. The operator needs rules. The route needs barriers. The organization needs to know who is responsible when the machine moves something it should not.

Agents create the same distinction for knowledge work.

A workflow is **agent-operable** when an agent can perform an explicitly defined share of the work with the data, authority, controls, evidence, and human handoffs required to make the result safe, repeatable, and answerable.

That definition matters because it moves the question away from “Can the model do this task?” and toward five more useful questions:

- Can the agent obtain the right context in a form it can interpret reliably?
- Is the real workflow mapped, including exceptions and tacit decisions?
- Which actions should the agent read, draft, decide, or execute?
- Under whose authority does it act, and within what boundary?
- Can the organization reconstruct what happened and assign accountability?

Agent operability is not a binary certification. It is a property of a particular workflow, for a particular agent, under particular conditions. The same agent may be operable for preparing an internal summary and inoperable for sending a customer commitment. The same workflow may be operable in one region and blocked in another because the systems, regulations, or ownership differ.

This is why enterprise-wide claims such as “we are agent-ready” are usually too vague to be useful. Operability lives at the level where work actually happens: the workflow, the action, the data object, the decision right, and the permission.

The category mistake behind most pilots

Most pilots are framed as software implementation projects. A team selects a model, connects several sources, adds a user interface, and measures output quality. But an agent is not merely another interface to the workflow. The agent becomes an actor inside it.

That changes the design object.

Classic UX asks: can a person complete the task efficiently? AUX asks: what will a person permit a system to do on their behalf, how does that permission expand or retract, and how does control return when the system is uncertain or wrong?

Agent operability extends the same logic to the enterprise operating model. It asks whether the organization itself is legible and accessible enough for a non-human actor to participate without borrowing invisible human infrastructure.

That invisible infrastructure includes:

- the analyst who knows which spreadsheet is actually current;
- the manager who recognizes that a “standard” exception is not standard in this market;
- the legal counsel who knows which clause can be softened without escalating;
- the employee’s identity, which silently carries years of accumulated permissions;
- the unrecorded conversation that explains why the documented process is not followed;
- the instinct to stop when something looks wrong, even if no rule explicitly says so.

Agents do not inherit this infrastructure automatically. It must be exposed, structured, bounded, and governed.

The three-layer thesis

This book organizes the problem into three layers:

1. **Data shape** — Can the agent consume the information and context the workflow depends on?
2. **Process design** — Can the work be decomposed into explicit actions, decisions, and handoffs at appropriate levels of autonomy?

3. **Trust & permissions** — Can the agent act under a distinct identity, within a bounded permission envelope, while leaving an evidence trail that makes its output answerable?

The three layers behave like a weakest-link system. High-quality data cannot compensate for an undefined approval boundary. A beautifully mapped process cannot compensate for inaccessible context. Strong identity infrastructure cannot rescue a workflow whose critical judgment remains undocumented.

> **Agent operability is not the average maturity of the three layers. It is the lowest maturity among them.**

The rest of this book builds the vocabulary and tools needed to work on that lowest layer deliberately.

CHAPTER 1

What IT leaders are actually saying

The public conversation about enterprise agents has changed.

Two years ago, the most common questions were about model capability: whether a model could reason over long documents, use tools, or produce acceptable work. Today, enterprise technology leaders increasingly describe a different bottleneck. The models are capable enough to expose how unprepared the surrounding organization is.

In July 2026, Box CEO Aaron Levie summarized themes from a dinner with leaders responsible for agent adoption in large enterprises. The recurring topics were not prompt engineering or model benchmarks. They were change management, the need to modernize operating models, and the unglamorous work of making structured and unstructured data usable by agents. The significance is not that one dinner discovered a universal truth. It is that the same pattern is now recognizable across organizations: the agent is often ready before the workflow is.

The consensus beneath the hype

Three ideas are converging.

1. Change management is not the rollout plan. It is part of the system design.

Traditional enterprise software implementation often separates the technical build from adoption. The system is configured; then training, communications, and change champions help people use it.

Agents blur that separation because adoption depends directly on how authority is designed. Employees are not merely learning a new tool. They are deciding whether to delegate work to a system that may remember, recommend, and act. Their trust depends on whether the agent's intent is visible, whether it asks before consequential actions, whether it can be corrected, and whether mistakes create recoverable or irreversible consequences.

A technically functioning agent with an unclear control model creates rational resistance. People are not "change resistant" because they refuse to approve invisible delegation. They are responding to an operating model that has not made the new relationship legible.

This is a central AUX principle: **trust is dynamic**. Autonomy expands with evidence and retracts with violation. Adoption therefore cannot be treated as a communication layer placed on top of the system. The rules by which trust is earned must be designed into the workflow.

2. Data readiness is really context readiness.

The enterprise data conversation was built around analytics, reporting, and machine learning. Agentic work imposes a different standard.

A dashboard can tolerate a field whose meaning is known by the analyst. A retrieval system can return a policy document without understanding whether it is superseded. A search result can expose five relevant sources and leave a human to reconcile them.

An acting agent needs more. It needs the right context for the specific decision, at the right moment, with enough provenance and structure to know what can be trusted. It needs to distinguish policy from commentary, current from obsolete, global from market-specific, approved from draft, and instruction from evidence.

This is why "we have the data" is often true and insufficient. The information exists, but its operational meaning is carried by folders, naming conventions, social knowledge, inbox history, and human interpretation.

The real prerequisite is not merely access to data. It is **shaped context**.

3. IT is becoming the central nervous system of the operating model.

For decades, many business workflows could be changed locally. A team adopted a new application, redesigned a process, or added automation within its function. Agents cut across those boundaries because useful work often requires context and action from several systems at once.

A customer onboarding agent may need CRM data, contract terms, identity verification, billing status, product configuration, support history, and communications. A campaign QA agent may need brand policy, market rules, asset libraries, media plans, analytics, and publishing access. The workflow is commercial, but its operability depends on identity, integration, policy enforcement, logging, and lifecycle management.

That moves IT from support function to operating-model architect. It also makes the old split between “business process” and “technical architecture” increasingly artificial. The agent experiences both as one environment.

Everyone agrees on the problem; few share a vocabulary for the solution

The enterprise market has many mature vocabularies around adjacent problems:

- data quality and data governance;
- business process management;
- identity and access management;
- cybersecurity and zero trust;
- model evaluation and responsible AI;
- automation and orchestration;
- user experience and change management.

Agent adoption touches all of them but belongs fully to none of them. This creates coordination problems.

The data team says the source is available. The process owner says the workflow is documented. Security says the system can use a service account. The AI team says the model passes evaluation. Yet nobody can answer whether the workflow is safe to delegate at the action level.

Agent operability provides a shared object for those teams to inspect together.

It does not replace existing disciplines. It connects them around the specific question: **what must be true for this agent to perform this part of this workflow under real enterprise conditions?**

From “AI use case” to “delegation case”

The phrase “AI use case” is too broad. It groups together a chatbot that summarizes a document and an agent that changes a production record, even though the operating requirements are entirely different.

A better framing is the **delegation case**.

A delegation case specifies:

- the workflow and business outcome;
- the actions being delegated;
- the human and system actors involved;
- the context required for each action;
- the maximum authority granted;

- the conditions under which the agent must stop or escalate;
- the evidence required after the action;
- the owner accountable for the agent's lifecycle and access.

This framing changes portfolio decisions. Instead of ranking ideas by excitement or theoretical time saved, leaders can rank delegation cases by value, operability, and heat.

The best first workflow is often not the most impressive one. It is the workflow where the organization can learn the operating discipline without placing trust under maximum load.

The emerging enterprise failure pattern

Across agent pilots, the same sequence appears:

1. A team demonstrates that the model can perform a meaningful cognitive task.
2. The pilot is celebrated as proof that the workflow can be automated.
3. Production requirements expose missing data semantics, undocumented exceptions, and cross-system permissions.
4. The team adds human approvals everywhere to get through review.
5. The workflow becomes slower or more cumbersome than the original.
6. Users stop trusting or using it.
7. The organization concludes that agents are not ready.

The wrong conclusion is drawn from the right evidence.

The agent may be ready. The workflow is not operable.

Agent operability makes that diagnosis actionable. It separates model limitations from organizational limitations, and it shows which layer must move before more engineering effort will create value.

Questions for the leadership team

- Are our pilots measuring model quality, or production operability?
- Which workflow failed because of the model, and which failed because of context, process, or permissions?
- Do we have one vocabulary that IT, security, legal, operations, and the business can use to discuss delegation?
- Is anyone explicitly accountable for trust and adoption, not only technical delivery?
- Are we modernizing workflows for agents, or placing agents inside workflows built for humans with browsers?

CHAPTER 2

The three layers of agent operability

THE AGENT OPERABILITY MODEL

01	02	03
DATA SHAPE	**PROCESS DESIGN**	**TRUST + PERMISSIONS**
Can the agent consume authoritative context?	What should it read, draft, decide, or execute?	As whom does it act, and who answers?

THE WEAKEST LAYER SETS THE MAXIMUM SAFE AUTONOMY

Agent operability is the capacity of a workflow to be performed by an agent with bounded autonomy, usable context, explicit decision rights, and reconstructable accountability.

The model has three layers:

Layer	Core question	Typical failure
1. Data shape	Can the agent consume the context required to do the work correctly?	Information exists but is ambiguous, stale, inaccessible, or dependent on tribal knowledge.
2. Process design	What should the agent read, draft, decide, or execute — and where must a human intervene?	The documented process omits exceptions, judgment, and real handoffs.
3. Trust & permissions	As whom does the agent act, within what boundary, and who answers for the result?	Borrowed logins, over-broad service accounts, unclear authorship, and no durable evidence trail.

A weakest-link system

Organizations often invest in the layers unevenly.

Layer 1 receives the most attention because data programs already exist. Layer 2 is improvised through workshops and automation diagrams. Layer 3 is postponed until security review, when it becomes a blocker rather than a design input.

But the layers do not add together like points in a maturity score. They multiply — and a near-zero score in one layer collapses the delegation case.

A useful mental model is:

> **Operability = minimum(Data Shape, Process Design, Trust & Permissions)**

This is not a mathematical claim. It is an operating rule. The weakest layer sets the maximum safe autonomy.

Consider a vendor compliance workflow:

- The agent has perfect access to policies and prior responses.
- The process is mapped with clear exception paths.
- The agent runs under a generic integration account with access to all vendors and no named sponsor.

The workflow is not highly operable. Its permission model caps it.

Now reverse the problem:

- The agent has a unique identity and narrow access.
- Every action is logged and reviewable.
- The policy repository contains duplicate, conflicting, and obsolete documents.

The workflow is still not highly operable. Its data shape caps it.

Layer 1 — Data shape

Data shape asks whether the information the workflow depends on is prepared for action, not merely available for reading.

This includes:

- **Findability** — Can the relevant source be located deterministically?
- **Authority** — Can the agent distinguish the source of truth from drafts and commentary?
- **Structure** — Are critical fields, states, and relationships explicit?
- **Semantics** — Are terms and values defined consistently?
- **Provenance** — Can the agent show where a fact, rule, or recommendation came from?
- **Freshness** — Can it identify whether the context is current enough for the action?
- **Scope** — Can it distinguish user, team, region, client, and enterprise boundaries?
- **Actionability** — Can the context drive a permitted tool call or decision without hidden human interpretation?

The final dimension is what separates readable from operable.

A policy PDF is readable. A policy clause with an owner, jurisdiction, effective date, supersession status, exception rule, and linked approval path is operable.

Layer 2 — Process design

Process design asks what the work actually is.

Most enterprise workflows have at least three versions:

1. the process in the official documentation;
2. the process leaders believe is followed;
3. the process experienced employees actually run.

Agents meet version three.

An agent-operable process map must capture:

- triggers and success conditions;
- human, system, and agent actors;
- inputs, intermediate artifacts, outputs, and audit artifacts;
- decision points and the evidence used at each one;
- routine paths, exceptions, dead ends, retries, and escalation paths;
- action heat and required control posture;
- ownership of each handoff;
- measurable quality, trust, and economic outcomes.

The central design choice is delegation. Every action must be placed deliberately across a spectrum from observation to autonomous execution.

auxfirst expresses this through two related models:

- an **autonomy ladder** — observe, recommend, draft, execute within a gate, or act autonomously;
- the **Action Heat Ladder** — the consequence of an action if it goes wrong, scored across reversibility, blast radius, exposure, commitment, and authority.

Autonomy should fit heat. It should not simply increase because the model becomes more capable.

Layer 3 — Trust & permissions

Trust & permissions asks whether the agent has a legitimate place in the organization's system of identity, authority, and accountability.

A production agent needs more than credentials. It needs an operating contract.

At minimum, the organization must define:

- a distinct identity or explicitly governed representation;
- a human sponsor accountable for lifecycle and access;
- an owner accountable for behavior and outcomes;
- a permission envelope specifying tools, data, actions, limits, and conditions;
- an autonomy posture per action class;
- escalation triggers and recipients;
- a memory policy defining what persists, where, and for how long;
- an authorship layer binding outputs to sources, agent version, policy, and human approval;
- an audit trail sufficient to reconstruct consequential actions;
- a review and expiry cadence.

The trust layer is not equivalent to security. Security protects systems and information. Trust architecture makes delegated action legible and answerable to people.

An agent can be secure in the narrow technical sense and still be untrustworthy in operation. It may authenticate correctly while acting beyond the user's intent. It may access only permitted records while using stale context. It may produce an accurate draft whose authorship and approval status are unclear.

Trust emerges from the combination of capability, boundaries, evidence, control, and accountability.

The three-layer assessment

A practical assessment begins with one workflow and one target delegation case.

For each layer, ask:

Data shape

- What information is required at each step?
- Which source is authoritative?
- Which meaning is still carried by folder location, naming convention, or human memory?
- What context may the agent retrieve but not retain?
- What data can inform a draft but cannot justify an execution?

Process design

- What happens in reality, including exceptions?
- Which steps are mechanical, judgmental, or political?
- Which actions are reversible?
- Where does exposure, commitment, or authority increase?
- What is the minimum useful autonomy for the business case?

Trust & permissions

- As whom does the agent act?
- What is the narrowest permission envelope that supports the delegation case?
- Who sponsors the identity and reviews its access?
- What must be visible before a human approves?

- What evidence must remain after the action?

The Agent Operability Map

The output should not be a generic maturity score. It should be a map of the workflow showing where operability breaks.

A useful map contains four views:

1. **Current-state workflow** — how the work actually runs today.
2. **Agent-ready workflow** — the target human–agent operating model.
3. **Layer gaps** — data, process, and trust constraints attached to specific steps.
4. **Sequenced fixes** — the smallest set of changes required to unlock the next safe level of autonomy.

This is deliberately narrower than an enterprise AI strategy. The purpose is not to describe everything the organization should become. It is to make one delegation case buildable, governable, and economically defensible.

A quick example: invoice exception handling

Imagine an agent that reviews invoices, matches them to purchase orders, and routes exceptions.

Data shape gap: supplier names are inconsistent, purchase orders are missing line-level references, and approval limits are stored in a spreadsheet maintained by one finance manager.

Process design gap: “routine discrepancy” is not defined. Experienced analysts use different thresholds based on supplier history and urgency.

Trust & permissions gap: the agent can read invoices and purchase orders but would need authority to change payment status and contact suppliers. No dedicated identity or communication approval model exists.

The first operable version should not autonomously resolve exceptions. It should:

- read and reconcile the records;
- classify discrepancy types;
- draft a recommended resolution with evidence;
- route high-heat cases to named approvers;
- leave payment status unchanged;
- log the sources, rules, and human decision.

This is less autonomous than the original vision and more valuable than a stalled pilot. Operability often advances by **cooling the action**, not by forcing the organization to accept more risk.

The principle to remember

> **Do not ask how autonomous the agent can be. Ask what level of autonomy the workflow has earned.**

CHAPTER 3

Data shape: readable is not operable

Enterprises contain enormous amounts of machine-readable information. That does not mean agents can use it to operate workflows.

The distinction begins with a simple observation: humans read with context they do not notice they are supplying.

An experienced employee opens a file and immediately recognizes that it is old. They know that “final” does not mean final in this team. They understand that a certain customer category requires a different approval path. They see an empty field and infer that the value lives in an email thread. They know which manager’s comment should be treated as a decision and which is merely a suggestion.

The document did not contain all of that meaning. The employee brought it.

Agents expose this gap because they require the organization to externalize enough context for the action to be justified and repeatable.

Five levels of data shape

The following maturity ladder is designed for workflow operability, not for general data governance.

Level	Description	What the agent can safely do
D0 — Human-located	Information exists across screens, inboxes, folders, and memory.	Observe only with heavy human guidance.
D1 — Machine-retrievable	Sources can be searched or fetched, but authority and meaning remain ambiguous.	Summarize and surface candidates.
D2 — Machine-interpretable	Key fields, taxonomies, versions, and relationships are explicit.	Classify, compare, and draft with cited evidence.
D3 — Action-linked	Context is connected to rules, states, and permitted next actions.	Recommend and execute bounded, reversible steps.
D4 — Governed context	Provenance, freshness, scope, retention, and policy are continuously enforced.	Operate repeatably within a defined permission envelope.

The maturity required depends on the heat of the action. A low-risk internal summary may be acceptable at D1. A customer commitment or financial action may require D3 or D4.

Structured and unstructured is the wrong binary

The common distinction between structured and unstructured data is useful but incomplete.

A database field can be structured and operationally meaningless. A well-governed policy document can be unstructured and highly operable. The important questions are not only how the information is stored, but whether the agent can determine:

- what the object is;
- who owns it;
- whether it is current;
- which scope it applies to;
- how it relates to other objects;
- which claims it supports;
- what action it permits or prohibits;
- what happens when sources conflict.

This is why data shape includes both technical structure and organizational semantics.

Tribal knowledge is invisible infrastructure

Tribal knowledge is often discussed as a documentation problem. For agents, it is more than that. It is hidden execution logic.

Examples include:

- “We never send that clause to public-sector clients.”
- “This supplier is allowed a five-day exception because of the old master agreement.”
- “If the customer mentions migration risk, bring in the solution architect before pricing.”
- “The dashboard shows revenue by booking date, but finance evaluates it by delivery date.”
- “Do not escalate small discrepancies in the final three days of the quarter unless tax is affected.”

These statements change actions. If they remain in people’s heads, the agent either fails or forces a human to reinsert the missing context at every run.

The goal is not to document every tacit nuance before starting. That would create an endless knowledge-management program. The goal is to identify **load-bearing judgment**: the hidden knowledge whose absence changes a consequential action.

Decision residue

One of the most valuable sources of context is the residue of prior decisions.

A policy tells the agent what should happen in general. Historical decisions show how the organization interprets the policy under real conditions.

But raw history is dangerous. Past decisions may be inconsistent, biased, obsolete, or based on facts no longer visible. The agent should not blindly imitate them.

The useful pattern is to transform selected decisions into governed examples containing:

- the situation and relevant facts;
- the rule or policy considered;
- the decision made;
- the named decision owner;
- the rationale;
- the scope and date;
- whether the example is still approved for reuse;
- the conditions under which it should not be generalized.

We call this **decision residue**: structured traces of how judgment was applied, retained as context without pretending that precedent is universal policy.

The authority problem

Retrieval systems often optimize relevance. Operability requires authority.

The most semantically similar document is not necessarily the one that should govern the action. Agents need an authority model that can distinguish:

- policy from guidance;
- approved from draft;
- current from superseded;
- global rule from local variation;
- legal requirement from internal preference;
- source evidence from generated interpretation.

A practical authority model can be simple. Every critical source should carry metadata for owner, status, effective date, scope, supersession, and review date. The agent should be instructed — and technically constrained where possible — to prioritize sources according to that hierarchy.

When authority conflicts, the correct behavior is usually not “choose the most likely answer.” It is “surface the conflict and escalate.” Graceful uncertainty is an AUX heuristic because confident resolution of ambiguous authority is one of the fastest ways to destroy judgment trust.

Context efficiency: more is not safer

A common response to missing context is to provide the agent with everything.

This can reduce quality rather than improve it. Irrelevant or conflicting context increases the chance that the agent will focus on the wrong source, blend scopes, or carry outdated information into the decision.

Context should be treated as an allocation problem. The agent needs the minimum sufficient context for the current action, selected according to explicit relevance and authority rules.

Useful controls include:

- task-specific retrieval rather than broad repository access;
- source allowlists for high-heat actions;
- maximum age rules;
- jurisdiction and client filters;
- separation between instruction, policy, evidence, and examples;
- context manifests showing what was included and excluded;
- memory boundaries preventing one case from contaminating another.

This is **context efficiency**: using context intelligently, not exhaustively.

From document to operable knowledge object

Consider a pricing policy stored as a PDF.

The PDF may be sufficient for a human to read. To become operable, the critical parts could be represented as a knowledge object with:

- policy ID and version;
- owner and approver;
- effective and review dates;
- applicable products, markets, and customer segments;
- standard discount bands;
- exception thresholds;
- required approver by threshold;
- prohibited combinations;
- linked evidence and original document;
- tool action permitted at each band;
- escalation target when the request does not fit.

The PDF remains the authoritative source. The knowledge object makes its operational meaning available to the workflow.

Data shape fixes should follow the delegation case

Do not begin by “cleaning all the data.” Begin with the actions the agent is expected to perform.

For each action, identify:

1. the minimum context required;
2. the authority level required;
3. the fields or semantics currently missing;
4. the cost of being wrong;
5. the smallest shaping intervention that makes the action safe enough.

This keeps the work bounded. A three-week operability audit should not become a multi-year data transformation. It should produce a prioritized fix list tied directly to the workflow and the level of autonomy being pursued.

Before moving on, ask

- Can you name the authoritative source for every consequential decision in this workflow?
- Does load-bearing tribal knowledge exist only in people's heads?
- Can the agent cite the evidence behind what it recommends or does?

The full scored assessment for this layer — 12 items, aligned with the audit methodology — is in **Appendix A**.

The principle to remember

> **An agent does not need all your data. It needs the right context, with enough authority and structure to justify the next permitted action.**

CHAPTER 4

Process design: the Action Heat Ladder

A workflow is not a list of tasks. It is a chain of actions with different consequences.

The distinction matters because agents are often discussed at the capability level: “the agent can handle onboarding” or “the agent can manage RFPs.” Those phrases hide the actions that create risk. Reading an RFP, extracting requirements, drafting an answer, selecting a commercial commitment, and submitting the response are not one capability. They are five different action classes with five different control requirements.

Agent process design begins by breaking the workflow into verbs and objects.

- read policy;
- retrieve customer history;
- classify requirement;
- draft response;
- recommend exception;
- approve discount;
- send commitment;
- update system of record.

Once the actions are visible, the team can decide what should move to the agent and at what temperature.

Four modes of work

For operability design, every workflow step can be placed into one of four plain-language modes:

Read

The agent observes, retrieves, summarizes, classifies, or monitors. It does not change the state of the business.

Read actions are usually the coolest, but not always harmless. Reading can still expose sensitive information, cross scope boundaries, or create privacy risk. Read-only is a permission posture, not a guarantee of safety.

Draft

The agent creates a proposed artifact for a human to review: an email, response, plan, record update, code change, or recommendation.

Drafting is one of the most powerful de-escalators. It captures much of the productivity value while keeping exposure and commitment at zero until a human acts.

Decide

The agent selects among options, applies a rule, prioritizes, routes, approves, or refuses.

Decision actions require evidence and explicit boundaries. A decision can be internal and reversible yet still materially shape outcomes. The key question is whether “right” can be checked and whether the decision falls inside a defined policy envelope.

Execute

The agent changes state: sends, publishes, pays, deletes, deploys, signs, grants, books, or commits.

Execution is where the enterprise experiences the full consequence of agency. Every execution action needs an identity, authority, limits, evidence, and a recovery design.

These modes are not a maturity ladder where execute is always the goal. The correct design may permanently keep a step at draft or decide. The objective is not maximum autonomy. It is the best human–agent allocation for the stakes.

Heat decides who pulls the trigger

auxfirst's **Action Heat Ladder** turns the vague question "is this risky?" into a repeatable reading.

Every agent action is scored across five dimensions from 0 (cool) to 4 (hot):

Dimension	Question	Cool end	Hot end
Reversibility	Can we take it back?	Fully undoable	Permanent or unrecoverable
Blast radius	How far does it spread?	One item	Whole estate or population
Exposure	Who sees it?	Stays inside the working loop	Public, customer, regulator, press
Commitment	What does it bind us to?	No promise or value movement	Legal, financial, or reputational commitment
Authority	What is it allowed to touch?	Read-only or sandbox	Production, identity, and access control

An action is as hot as its hottest dimension. Do not average the scores.

A routine invoice email may be highly reversible and narrow in scope, but it leaves the organization and requests payment. Exposure and commitment make it a high-heat action. Four cool dimensions do not cancel one hot dimension.

Five control postures

Heat should change who pulls the trigger.

Heat band	Default control posture	Meaning
Low	Auto-run	The agent may execute and log.
Low–medium	Auto-run with sampled review	The agent executes within limits; humans audit a sample.
Medium	Propose + approve	The agent prepares the action; a human approves the commit.
High	Named approver	A specific accountable role must authorize each action.
Critical	Human executes	The agent may assist, but a human performs the action in the authoritative system.

The bands are not universal laws. They are a starting policy. Regulated environments, vulnerable populations, or high-value transactions may require stricter controls.

Cooling actions instead of blocking workflows

High heat does not automatically mean “do not use an agent.” It means redesign the action.

Six common de-escalators are especially useful:

1. **Draft-only output** — the agent produces; a human sends or commits.
2. **Sandboxed environment** — the agent acts on test data or in staging.
3. **Undo window** — delayed send, soft delete, versioned writes, or reversible state.
4. **Hard caps** — spend limits, rate limits, quantity limits, and scope ceilings.
5. **Sampled review** — a human audits a defined share on a cadence.
6. **Narrow allowlist** — the agent touches named records or action classes, not an entire category.

This is the practical route to safe autonomy. The team does not debate whether agents are trustworthy in the abstract. It changes the action until the required trust fits the evidence available.

Worked example: an RFP response workflow

RFP response is a useful example because it combines data retrieval, judgment, cross-functional access, commercial commitment, and external communication.

Current-state reality

A sales lead receives the RFP and forwards it to a bid manager. The bid manager creates a folder, copies an old response, and asks subject-matter experts for input. Security answers a questionnaire from its own library. Legal reviews clauses. Finance approves pricing. An executive approves non-standard commitments. The bid manager reconciles versions and submits through a portal.

The official process may show six steps. The real process contains dozens of micro-decisions:

- Is the opportunity worth pursuing?
- Which prior response is safe to reuse?
- Has the policy changed since the prior answer?
- Does the requirement ask for a current capability or a roadmap promise?
- Which claims require evidence?
- Which clause creates a non-standard obligation?
- Is the price inside the approved corridor?
- Who can approve an exception before the deadline?

Step 1 — Read and decompose

Agent actions: ingest RFP, classify requirements, identify deadlines, map questions to owners, and flag missing artifacts.

Heat: low to low–medium. The actions are internal and mostly reversible, but the agent may access confidential documents.

Control: auto-run with logging and source restrictions.

Data-shape requirement: the RFP, response library, policy sources, owner directory, and current capability catalog must be retrievable and scoped.

Step 2 — Retrieve evidence

Agent actions: find approved claims, certifications, case studies, policy language, and prior answers.

Heat: low–medium. No external action occurs, but context leakage and stale evidence are material risks.

Control: auto-run from approved repositories; show provenance and freshness.

Design requirement: separate authoritative evidence from persuasive examples. The agent must never convert a roadmap item into a current claim.

Step 3 — Draft responses

Agent actions: produce answer drafts, cite sources, mark confidence, and identify questions that require human input.

Heat: medium. The output is not yet external, but it shapes a commercial artifact.

Control: propose + approve by question owner or bid manager.

Trust patterns: Confidence Cues, Progressive Transparency, and Loop In Experts. The draft should expose evidence without burying the reviewer in raw context.

Step 4 — Decide compliance posture

Agent actions: classify each requirement as compliant, partially compliant, planned, exception required, or cannot meet.

Heat: high for ambiguous requirements. The classification can create legal and commercial consequences even before submission.

Control: the agent may recommend; named owners decide exceptions.

Process requirement: the categories and evidence thresholds must be explicit. “Partially compliant” cannot be a stylistic judgment.

Step 5 — Construct commercial commitments

Agent actions: propose pricing, delivery assumptions, service levels, and contractual deviations.

Heat: high to critical because commitment and exposure are high.

Control: agent drafts; finance, legal, and executive approvers authorize according to thresholds. The agent does not independently sign or submit non-standard commitments.

Permission requirement: the agent may read approved pricing corridors and clause libraries but cannot change approval limits or create new contractual authority.

Step 6 — Submit

Agent actions: package files, validate completeness, and upload through the buyer portal.

Heat: high. Submission is external, often irreversible after the deadline, and binds the company to the response.

Control: named human approves the final package. Depending on policy, the agent may upload after approval or the human may execute the submission.

Authorship requirement: the package must record the agent version, evidence sources, question owners, approvals, and final submitter.

The human–agent operating model

The target process is not “the agent writes the RFP.” It is a designed allocation:

Workflow segment	Agent owns	Human owns
Intake	Decomposition, routing, deadline extraction	Pursuit decision

Workflow segment	Agent owns	Human owns
Evidence	Retrieval, source matching, freshness checks	Authority conflict resolution
Drafting	First drafts, citations, gap flags	Claims approval and nuance
Compliance	Recommendation against explicit criteria	Exceptions and material interpretation
Commercial	Scenario preparation inside corridors	Commitments, deviations, approval
Submission	Completeness validation, packaging, optional upload	Final authorization and accountability

This model is more useful than a percentage automation target. It makes the boundaries visible and gives the organization a path to expand autonomy later if evidence supports it.

Where human judgment is load-bearing

Teams often overestimate the amount of judgment in a workflow because experienced people perform routine pattern matching fluently. They also underestimate judgment because the process map labels a step “review.”

To distinguish load-bearing judgment from habit, ask:

- Does the step resolve ambiguity that is not captured in policy?
- Does it balance competing values or stakeholder interests?
- Does it create a legal, financial, ethical, or reputational commitment?
- Would reasonable experts disagree even with the same facts?
- Is the decision sensitive to context that is hard to observe or encode?
- Is the cost of a plausible but wrong decision high?

If the answer is no, the step may be mechanical and suitable for greater automation. If yes, the design should preserve human accountability or deliberately build a stronger decision policy and evidence base.

Runtime design: the forgotten process layer

A static process map is not enough. Agents run in time.

The runtime design should specify:

- what triggers the workflow;
- what counts as a successful end;
- timeouts and deadlines;
- retry limits;
- dead ends and escalation routes;
- which steps can be rerun safely;
- how duplicate actions are prevented;
- how the agent behaves when a system is unavailable;
- how a human pauses, resumes, or terminates the run;
- how state is reconciled after partial failure.

Many “model failures” in production are runtime failures: the agent retries a payment, sends a duplicate message, acts on stale state, or cannot recover after one tool call fails.

The principle to remember

> **Map actions, not aspirations. Heat each action. Then give the agent only the autonomy that the action and evidence can carry.**

CHAPTER 5

The permissions problem nobody owns

The permissions problem appears late because pilots borrow authority from humans.

A developer runs the agent using their credentials. A process owner uploads the files. A tester confirms every tool call. A shared integration account connects the systems. The demo works because the human operator quietly provides the identity, judgment, and access model.

Production removes that convenience. The agent must operate repeatedly, across users and cases, with access that is discoverable, reviewable, revocable, and proportionate to the work.

This is not solved by giving the agent "a service account." A service account is an implementation mechanism. Agent operability requires an identity model.

Why cross-functional workflows break human role models

Enterprise access is usually organized around people, applications, teams, and job roles.

An agent may not fit any one of them.

Consider a customer onboarding workflow. The agent may need to:

- read the signed contract;
- verify account and billing information;
- create records in CRM and product systems;
- request missing compliance documents;
- schedule implementation tasks;
- notify internal owners;
- monitor completion;
- escalate exceptions.

No single employee may hold all of these permissions. That is often intentional. Segregation of duties prevents one person from controlling the entire chain.

If the agent is granted the union of everyone's access, it can become more privileged than any human. If it acts under one user's identity, accountability and policy become misleading. If it asks humans to perform every protected step, the workflow may preserve all the friction the agent was meant to remove.

The answer is not to copy the human role model. It is to define the agent as a governed actor with its own bounded role.

The Agent Identity Card

The **Agent Identity Card** is the human-readable artifact that defines who the agent is inside the organization.

It should be understandable by security, IT, legal, the process owner, and the business sponsor. It is not a credential file. It is the agent's organizational identity and operating contract at a glance.

Core fields

Field	What it defines
Agent name and ID	Unique, discoverable identity.
Purpose	The business outcome and workflow the agent exists to support.
Sponsor	Human accountable for the identity lifecycle and access.
Product owner	Human accountable for behavior, roadmap, and outcomes.
Process owner	Human accountable for workflow truth and exceptions.
Scope	Business unit, region, customer class, and environments covered.
Data domains	Sources the agent may read, with restrictions.
Tools and systems	Named capabilities the agent may call.
Action classes	Read, draft, decide, and execute actions permitted.
Permission envelope	Limits, conditions, caps, and prohibited actions.
Trust stage	Functional, contextual, judgment, or advocacy trust earned.
Escalation path	Who receives which uncertainty or exception.
Memory policy	What persists, where, for how long, and editable by whom.
Authorship policy	How outputs disclose agent, evidence, and approval.
Audit and review	Log location, access review cadence, expiry, and rollback owner.

The Identity Card is deliberately one page. It creates a shared reference before technical teams encode the details into IAM, policy engines, tool manifests, and runtime configuration.

Example Agent Identity Card

Agent: RFP Response Agent · `AGT-BID-014` **Purpose:** Reduce response cycle time while preserving claim accuracy and approval accountability. **Sponsor:** VP Commercial Operations **Product owner:** Head of Bid Management **Process owner:** Director of Enterprise Sales **Environment:** EMEA enterprise bids; production response workspace only **Trust stage:** Contextual trust earned; judgment trust limited to low-ambiguity classifications

May read: approved policy library, certifications, product capability catalog, selected prior responses, opportunity record, approved pricing corridors. **May draft:** response text, compliance classification recommendation, evidence list, clarification questions, submission checklist. **May decide:** routing, question ownership, duplicate detection, low-risk requirement taxonomy. **May execute:** create workspace, assign tasks, update internal status, package approved documents. **May not:** approve exceptions, create non-standard commitments, change pricing, send external clarification, submit final response without named approval.

Limits: one opportunity workspace at a time; no cross-client memory; no access to HR or unrelated customer records; external communication disabled; all source documents must carry approved status. **Escalates to:**

Security for unsupported claims; Legal for clause deviation; Finance for pricing outside corridor; Bid Lead for authority conflict. **Review:** quarterly access review; automatic expiry after 180 days without sponsor renewal; immediate pause after high-severity trust incident.

This card gives the security team something concrete to review. It also exposes design gaps before implementation. If the team cannot complete a field, the workflow is not ready to grant that form of delegation.

The permission envelope

A permission is usually described as access to a resource: read this database, write that record, call this API.

An agent needs a richer construct because the same tool may be safe or unsafe depending on context.

The **permission envelope** is the bounded combination of:

- **resources** — which systems, objects, and data scopes;
- **actions** — which verbs the agent may perform;
- **conditions** — under what workflow state, user intent, or evidence threshold;
- **limits** — quantity, value, rate, time, region, and blast radius caps;
- **autonomy posture** — auto-run, sampled review, approval, named approver, or human execution;
- **duration** — how long the permission exists;
- **delegation chain** — whether the agent acts on its own authority or on behalf of a user;
- **escalation** — where the action routes when it exceeds the envelope;
- **evidence requirement** — what must be logged before and after execution.

A useful permission statement is therefore not:

> “The agent can update CRM records.”

It is:

> “The agent may update the `onboarding_status` field for accounts in the EMEA implementation queue after contract validation succeeds, for up to 50 accounts per run, with every change logged and any conflicting account state routed to the implementation owner.”

The second statement can be reviewed, encoded, tested, and audited.

As itself or on behalf of a user?

Agent identity systems increasingly distinguish two operating modes.

Agent’s own authority

The agent acts as a distinct principal. Its permissions are tied to its role and lifecycle.

This is appropriate for persistent process agents performing a defined organizational function: monitoring queues, reconciling records, preparing reports, or executing bounded operational actions.

Benefits include clearer accountability, stable least-privilege design, and access that does not depend on one employee remaining in role.

On behalf of a user

The agent acts within the user’s authorization context for a specific request.

This is appropriate when the action represents the user's intent and should not exceed the user's own rights: searching personal files, drafting within a user workspace, or initiating an action the user is authorized to perform.

The critical design question is whether the agent may combine its own organizational authority with user-delegated authority. That combination can create unexpected privilege. The delegation chain must remain visible in the audit trail.

Why "run it under Dave's login" fails

Borrowing a human identity creates several problems:

- the audit log attributes agent actions to the employee;
- permissions persist when the workflow purpose changes;
- the agent may inherit unrelated access accumulated over years;
- offboarding or role changes break the system;
- segregation of duties becomes unclear;
- incident response cannot distinguish human and software behavior;
- users cannot know whether an action was performed by Dave or by a system acting as Dave.

The convenience is temporary. The accountability debt is permanent.

Sponsors, owners, and approvers are different roles

Agent governance becomes vague when one person is described as "the owner." At least three accountabilities should be separated:

- **Sponsor:** accountable for why the agent exists, whether it remains authorized, and whether access should continue.
- **Product owner:** accountable for what the agent does, how it performs, and how it evolves.
- **Process owner:** accountable for the truth of the workflow, business rules, and exceptions.

A fourth role, the **trust owner**, may be explicit in larger programs. This person is accountable for adoption, autonomy boundaries, experience patterns, and whether trust is being earned or eroded.

Named human accountability does not mean humans approve every action. It means the organization knows who is responsible for the system's continued operation and for changing its boundaries.

Trust tier versus permission level

Trust and permission are related but not identical.

auxfirst's trust ladder has four stages:

1. **Functional trust** — can the agent complete basic tasks reliably?
2. **Contextual trust** — does it understand relevant history, preferences, and scope?
3. **Judgment trust** — can it make good calls under ambiguity and escalate appropriately?
4. **Advocacy trust** — will it act in the user's interest when incentives or objectives conflict?

An agent may have broad technical permission but low earned trust. That is a dangerous state. It may also have high demonstrated reliability but intentionally narrow permission because the action is high heat.

The Identity Card should record both:

- the **trust stage earned through evidence**;

- the **permission envelope granted for this workflow**.

The gap between them is informative. If permission exceeds earned trust, risk is being assumed. If trust greatly exceeds permission, the organization may have an opportunity to unlock value safely.

Access must have a lifecycle

Human access programs already recognize joiner, mover, and leaver events. Agents need equivalent lifecycle controls.

An agent identity should have:

- creation approval;
- a named sponsor;
- environment and scope assignment;
- time-bound permissions where possible;
- periodic access review;
- version and deployment linkage;
- inactivity expiry;
- incident-triggered suspension;
- retirement and credential revocation;
- retention rules for logs and memory after retirement.

The identity should not outlive the delegation case that justified it.

The principle to remember

> **An agent needs more than credentials. It needs a name, a sponsor, a bounded role, an expiry, and a trail that shows when it acted as itself and when it acted for someone else.**

CHAPTER 6

Trust architecture: making agent output answerable

Security asks whether the system is protected. Trust architecture asks whether delegated action can be understood, challenged, reconstructed, and owned.

The distinction becomes important when the actor is software.

A human-authored document carries familiar social signals. The author has a role, reputation, manager, and employment relationship. Colleagues may know how the person reached the conclusion. If the output is wrong, the organization has processes for correction and accountability.

Agent output can look equally polished while its origins are opaque. Which model produced it? Which sources were retrieved? What policy version applied? Was the agent acting under a user's instruction or on a schedule? Did a human approve it? Was the final artifact edited after approval? What uncertainty was suppressed?

Without answers, the output is difficult to trust even when it is accurate.

The authorship layer

The **authorship layer** is the metadata and evidence that binds an agent-produced artifact or action to its origin, authority, and approval state.

For a consequential output, it should make the following visible or reconstructable:

- the agent identity and version;
- the initiating user, event, or schedule;
- the purpose and workflow instance;
- the model and relevant configuration version;
- the sources and their authority status;
- the rules, policy, or trust contract applied;
- the confidence or uncertainty state;
- the human reviewers and approvals;
- the final executor or submitter;
- the time and immutable audit reference.

Not all of this belongs in the user interface. AUX uses **progressive transparency**: show more reasoning and evidence while trust is being established or when stakes are high, then taper the detail without removing the underlying record.

The authorship layer supports both experience and governance. A reviewer sees enough to make a decision. An auditor can later reconstruct the full chain.

Answerability is stronger than explainability

"Explainable AI" is often framed as making model reasoning understandable. That is useful but insufficient for enterprise action.

A plausible explanation does not prove that the correct data, authority, and approval process were used. Chain-of-thought-style narratives can be incomplete or misleading. The organization needs operational evidence, not only a story.

Answerability means the system can answer:

- What was the agent trying to do?
- What was it allowed to do?
- What context did it use?
- What did it know was uncertain?
- Which rule or policy governed the action?
- Who approved or overrode it?
- What changed in the business as a result?
- Who is accountable for the workflow and the agent?
- Can the action be reversed or compensated?

Answerability shifts trust from “the model sounded reasonable” to “the system left receipts.”

The trust contract

The **Trust Contract** is the agent’s operational constitution.

It defines:

- scope and purpose;
- autonomy levels by action class;
- prohibited actions;
- memory and retention policy;
- evidence requirements;
- escalation triggers and service levels;
- user control and escape hatches;
- guarantees the system is expected to maintain;
- named trust gaps that block release or execution;
- incident response and rollback posture.

At auxfirst, the Trust Contract sits across three altitudes:

1. **AUX Design** defines the relationship and the experience.
2. **TrustKit** expresses the rules as a shared, machine-readable standard.
3. **The Trust Harness** enforces the contract at runtime through policies, gates, telemetry, and escalation.

The principle is more important than any specific implementation: discipline without a spec becomes a poster; a spec without runtime enforcement becomes a document; runtime controls without a coherent trust model become theatre.

Agents cannot keep themselves secure

It is tempting to instruct an agent to “be secure,” “respect permissions,” or “ask before risky actions.” These instructions may influence behavior, but they are not security controls.

A probabilistic model should not be the final authority on whether it is authorized to act. The architecture around it must enforce:

- authentication and authorization;
- least privilege;
- tool input validation;
- data-scope isolation;
- rate and value limits;

- human confirmation for high-risk actions;
- logging and tamper-resistant records;
- anomaly detection;
- timeout, retry, and rollback policy;
- separation of instructions from untrusted content;
- signed or verifiable inter-agent communication where relevant.

The agent may propose. Deterministic policy decides whether the proposal is permitted.

This is the trust harness pattern: the model reasons inside a bounded runtime that can stop, transform, route, or reject actions.

Continuous conformance, not one-time approval

A security review captures one version of the system under one set of assumptions. Agents change more frequently than traditional applications:

- models are upgraded;
- prompts and policies evolve;
- tools are added;
- retrieval sources change;
- users discover new behaviors;
- memory accumulates;
- vendors alter APIs and commercial terms;
- workflows expand into new markets or data classes.

Trust conformance must therefore be continuous.

A practical operating cadence includes:

Pre-release gates

- validate the Agent Identity Card and permission envelope;
- run action heat checks for new capabilities;
- evaluate against golden cases and known failure modes;
- block high-severity heuristic violations;
- confirm logging and rollback behavior;
- verify source authority and data scope.

Runtime monitoring

- track action volumes by heat band;
- monitor escalation and override rates;
- detect unusual tool or data access;
- compare planned versus actual autonomy;
- sample low–medium heat actions;
- alert on repeated uncertainty, retries, or boundary pressure;
- record trust incidents using a named taxonomy.

Periodic review

- review permissions and sponsor status;
- re-score actions after workflow or policy changes;
- analyze incidents and near misses;
- promote or demote autonomy based on evidence;
- update examples, policies, and de-escalators;
- test whether the economic case still holds.

Trust is dynamic. The operating system must be capable of retracting autonomy as well as granting it.

The trust gap taxonomy

“The agent is weird” is not a useful incident category.

Naming failure modes creates operational leverage. auxfirst’s trust gap taxonomy groups failures across the four trust stages.

Examples include:

- **Functional:** hallucination, silent degradation, inconsistent output.
- **Contextual:** memory amnesia, ignored preference, context leakage.
- **Judgment:** overreach in ambiguity, confident nonsense, refusal when escalation is required.
- **Advocacy:** optimizing a platform metric over the user, incentive misalignment, loyalty leakage.

A named gap can be linked to severity, owner, fix pattern, test case, and release gate. Every repeated incident should improve the system for future workflows.

The evidence loop

A trustworthy operating loop looks like this:

1. The workflow supplies scoped context.
2. The Trust Contract defines the current permission and autonomy budget.
3. The agent proposes a plan or action.
4. Deterministic controls evaluate scope, heat, and policy.
5. The user or named approver steers or authorizes where required.
6. The agent executes within the envelope.
7. The system records evidence, outcomes, and exceptions.
8. Monitoring turns incidents and overrides into new policy, tests, and examples.

This is how trust architecture compounds. The organization does not merely collect logs. It converts operational experience into better boundaries.

What to measure

Quality alone cannot describe a production agent. A balanced operating view should include:

Quality metrics

- accuracy and completeness;
- evidence correctness;
- consistency;
- task success;
- error and rework rate.

Trust metrics

- approval rate by action class;
- override and correction rate;
- escalation appropriateness;
- boundary violations;
- unsupported confidence;
- user-reported trust incidents;
- autonomy promotion or demotion events.

Operational metrics

- cycle time;
- queue reduction;
- handoff latency;
- runtime failure and retry rate;
- cost per completed workflow;
- percentage of actions within the intended envelope.

Economic metrics

- hours returned;
- capacity released;
- revenue accelerated;
- loss or risk avoided;
- cost of controls and human review;
- value realized at the current autonomy level.

The CFO should see the net operating case, not only gross time savings. An agent that saves 1,000 hours but creates 900 hours of review and remediation has not transformed the workflow.

Trust incidents and recovery

Trust loss is not graceful. One high-heat violation can collapse confidence built through hundreds of routine successes.

The recovery design should exist before the incident:

- pause or kill switch;
- affected-workflow isolation;
- clear user notification;
- reconstruction from audit evidence;
- correction or compensating action;
- named incident owner;
- criteria for restoring operation;
- autonomy rollback;
- communication to affected stakeholders;
- new test or policy created from the incident.

An escape hatch is not only a button in the interface. It is an organizational capability to regain control.

The principle to remember

> **Trustworthy agents do not ask the model to police itself. They surround probabilistic judgment with deterministic identity, policy, evidence, escalation, and recovery.**

CHAPTER 7

The new operating roles

Agent adoption is often assigned to a committee and delivered by a project team. Neither structure is sufficient for sustained operation.

A committee can align stakeholders but rarely owns the behavior of a live agent. A project team can launch a pilot but often dissolves when the system enters production. Agentic work needs durable, cross-functional ownership because the agent continues to learn, encounter exceptions, and press against its boundaries after launch.

The organizational question is not “Who owns AI?” It is “Who owns this agent, this workflow, this trust relationship, and this access lifecycle?”

The internal FDE pattern

The Forward Deployed Engineer, or FDE, emerged as a pattern for bringing technical builders close to the customer’s real environment. In enterprise agent adoption, a similar role is needed internally.

The internal FDE is not simply an engineer embedded in a business unit. It is a translator and system builder who can move across process, data, tools, policy, and user behavior.

This person or small pod works where the abstractions break:

- observes the workflow as it actually runs;
- identifies hidden judgment and exception paths;
- translates business outcomes into agent actions and evaluation criteria;
- shapes data and context for the workflow;
- designs tools, permissions, and escalation paths with security;
- prototypes the human–agent operating model;
- watches the first production runs;
- converts incidents into durable policy and product improvements.

The role is “forward deployed” because it sits inside the operating reality rather than waiting for requirements to arrive in a central AI team.

The agent operability pod

The most effective unit is usually a small pod aligned to one workflow or domain.

Role	Primary accountability
Executive sponsor	Mandate, funding, and organizational air cover.
Agent product owner	Agent behavior, roadmap, outcomes, and improvement cadence.
Process owner	Workflow truth, exception logic, and business acceptance.
Internal FDE / agent architect	End-to-end operability design and technical translation.
Platform / engineering owner	Runtime, integrations, reliability, and deployment.

Role	Primary accountability
Security / identity owner	Identity, access, policy enforcement, and incident controls.
Risk / legal owner	Regulatory, contractual, and accountability requirements.
AUX / trust owner	Adoption, autonomy boundaries, transparency, control, and trust measurement.

Not every organization needs eight separate people. Several roles may be held by the same person. What matters is that the accountabilities are explicit and no critical perspective is absent.

Why IT-only ownership fails

IT can build the system but cannot define the business truth alone.

When the business does not actively own the workflow, the agent automates the process as documented rather than as performed. Exceptions are discovered after launch. Business owners treat errors as technical failures even when the underlying rule was never clear. Adoption becomes someone else's problem.

The inverse also fails. A business-led agent built without identity, architecture, and security ownership becomes a shadow system with fragile credentials and no path to scale.

Agent operability is inherently cross-functional because the agent crosses functional boundaries in the act of doing work.

The trust owner

Many organizations have product owners, process owners, data owners, and risk owners. Few have a person accountable for whether the relationship between users and agents remains trustworthy.

The trust owner focuses on questions such as:

- Is the agent's intent visible before consequential action?
- Does autonomy match the evidence it has earned?
- Are users able to steer, override, and recover?
- Are confidence and uncertainty communicated appropriately?
- Are escalation paths working in practice?
- Is memory accurate, scoped, and editable?
- Are repeated trust gaps becoming policy and tests?
- Is adoption rising because trust is earned, or because use is mandated?

This role can sit within product, design, AI governance, or operations. The title matters less than the accountability.

From centre of excellence to operating network

A central AI team is useful for shared platforms, standards, vendor management, and reusable controls. It becomes a bottleneck when every workflow depends on it for local process knowledge and decisions.

A scalable model has two levels:

Central enablement

- identity and agent registry;
- approved models and runtimes;
- tool and MCP standards;
- TrustKit, policy templates, and audit schemas;
- evaluation infrastructure;
- observability and incident response;
- reusable patterns and de-escalators;
- vendor and commercial governance.

Domain operability pods

- workflow mapping;
- data shaping;
- business rules and exceptions;
- Agent Identity Cards;
- action heat assessments;
- user testing and adoption;
- local operating metrics;
- incident learning and improvement.

The centre creates the paved road. The pods make real workflows operable on it.

The new remit of the technology function

When automation could touch only a minority of work, IT could remain primarily an infrastructure and application function. Agents can touch almost every form of knowledge work. That expands the remit.

Technology leadership increasingly becomes responsible for the organization's **delegation architecture**:

- how non-human actors are identified;
- how context moves across systems;
- how decision rights are expressed in policy;
- how tools expose safe action surfaces;
- how agent activity is observed;
- how human accountability remains intact;
- how workflows evolve as autonomy expands.

This does not make IT the owner of every business decision. It makes IT the architect of the environment in which delegated decisions can occur safely.

Skills the organization must build

Agent operability draws on familiar skills in new combinations:

- process discovery that captures exceptions rather than idealized flows;
- context and knowledge engineering;
- identity and policy design for non-human actors;
- evaluation of probabilistic behavior;
- human–agent interaction and trust design;
- runtime and observability design;
- economic modeling of supervised automation;
- incident analysis that connects model, process, data, and authority.

The scarce capability is not always deep model expertise. It is the ability to see the whole operating system around the model.

A practical ownership test

For any live or planned agent, ask five people independently:

1. Who can pause the agent immediately?
2. Who approves a new tool or data source?
3. Who decides whether autonomy may expand?
4. Who owns an incorrect but policy-compliant decision?
5. Who reviews the agent's identity and access when the sponsor changes role?

If the answers differ, the ownership model is not yet operable.

The principle to remember

> **Agents need product ownership, process ownership, identity ownership, and trust ownership. A committee can advise all four; it cannot substitute for them.**

CHAPTER 8

What headless means for your vendor stack

Enterprise applications were designed around human users navigating interfaces. Agents prefer tools, schemas, events, and explicit contracts.

This does not mean every application must remove its interface. It means the useful capabilities of the application must be available without requiring a human to click through the interface at every step.

That is the practical meaning of **headless** for the agentic enterprise.

From system of engagement to system of action

A traditional enterprise application exposes:

- screens;
- forms;
- dashboards;
- menus;
- user roles;
- exports;
- workflow rules hidden in configuration.

An agent-operable application also exposes:

- stable APIs or tool interfaces;
- machine-readable schemas and state;
- event triggers and webhooks;
- granular authorization scopes;
- idempotent and reversible operations where possible;
- preview or dry-run modes;
- rate, value, and scope limits;
- audit logs that identify non-human actors;
- versioned documentation;
- clear commercial rights for automated use.

The agent should not need to impersonate a human operating the screen unless no safer interface exists. Browser automation can be valuable as a bridge, but it is brittle, difficult to govern, and often tied to human credentials.

APIs are necessary and insufficient

A vendor may say “we have an API.” That does not mean the product is agent-operable.

The API may expose only a fraction of the workflow. Permissions may be all-or-nothing. Documentation may be incomplete. Write operations may not support previews or idempotency. Rate limits may make the economic case impossible. The vendor may prohibit automated use in its terms. Audit logs may attribute every call to one integration account.

Headless readiness has four dimensions.

1. Technical operability

- Can the agent read the required objects and states?
- Can it perform the required actions through documented interfaces?
- Are schemas stable, typed, and versioned?
- Are events available, or must the agent poll?
- Are operations idempotent or safely retryable?
- Can the system return structured errors the agent can route?

2. Security operability

- Can the agent authenticate as a distinct identity?
- Are scopes granular by action and object?
- Can access be time-bound and reviewed?
- Does the vendor support delegated and agent-owned authority?
- Can sensitive operations require additional approval?
- Are tool calls and data access logged with sufficient detail?

3. Process operability

- Can the application expose the actual business state, not only records?
- Are approval, exception, and rollback states available?
- Can the agent preview consequences before commit?
- Can humans pause, correct, and resume the workflow?
- Can the system distinguish draft, approved, and executed artifacts?

4. Commercial operability

- Do the terms permit agent access and automated action?
- Is usage priced per user, call, token, workflow, or outcome?
- Does an agent require a human seat?
- Are there restrictions on caching, training, retention, or derived data?
- Who owns agent-generated outputs and telemetry?
- Can the vendor change limits or capabilities without sufficient notice?

Commercial operability is often ignored until scale. A technically elegant agent can become uneconomic if every non-human actor requires a premium seat across ten applications.

MCP, tools, and the new interoperability layer

The Model Context Protocol and similar tool standards make it easier for agents to discover and call capabilities across applications. This is strategically important because it separates the agent experience from the vendor interface.

But interoperability does not remove governance. A tool description is not a permission model. A connected server can expose actions with very different heat. Clients and servers still need authorization, input validation, confirmation for sensitive operations, output validation, rate limiting, and audit logs.

The enterprise needs a **tool catalogue** that records:

- tool owner and vendor;
- action classes exposed;
- required identity and authorization mode;
- data domains touched;
- heat ceiling;
- confirmation policy;

- rate and value limits;
- version and change history;
- evaluation and incident status;
- approved agents and workflows.

This is the headless equivalent of an application catalogue, but oriented around verbs and authority rather than software titles.

The renewal negotiation changes

Vendor renewal used to focus on seats, storage, service levels, integrations, and discounts. Agent operability adds a new set of questions.

Questions for your next renewal

1. Can non-human agents have first-class identities distinct from users and generic integrations?
2. Can permissions be scoped by action, object, field, region, environment, and value threshold?
3. Which capabilities are available through API, MCP, events, or other tool interfaces?
4. Which high-value actions still require browser interaction?
5. Are write operations previewable, idempotent, reversible, and auditable?
6. Can human approval be inserted without leaving the workflow state ambiguous?
7. Do audit logs show agent identity, delegated user, action, inputs, result, and policy decision?
8. What data may the vendor retain or use from agent interactions?
9. How are API and tool changes versioned and communicated?
10. What are the pricing and licensing rules for agents, service identities, and automated volume?
11. Can we export the agent's operational data, policies, and audit evidence if we leave?
12. Will the vendor contractually support our required permission and evidence model?

These questions shift leverage. The best enterprise application is no longer only the one employees prefer to use. It is the one that can participate safely in a multi-agent operating environment without trapping context, identity, or evidence.

The agent-operable vendor review

Review each area against evidence from the product, architecture, contract, and operating experience. Do not collapse the result into a single maturity score: one critical identity, permission, or auditability gap can set the autonomy ceiling for the workflow.

Area	Questions to verify	Evidence of operability
Interfaces	Are consequential actions exposed through stable, documented interfaces rather than only the human UI?	A complete, versioned action surface with clear input, output, and error semantics.
Identity	Can non-human actors have distinct identities, sponsors, lifecycles, and delegated authority?	First-class agent or workload identity with attributable delegation.
Permissions	Can access be constrained by action, object, field, region, environment, time, and value?	A conditional, time-bound permission envelope rather than an all-or-nothing role.
Runtime safety	Can sensitive actions be previewed, limited, approved, retried safely, and reversed?	Dry runs, idempotency, caps, approval hooks, rollback, and policy enforcement.

Area	Questions to verify	Evidence of operability
Auditability	Can the enterprise reconstruct who or what acted, under whose authority, using which evidence and policy?	An action-level trail linking agent identity, delegated user, inputs, result, evidence, and policy decision.
Commercial fit	Are automated use, service identities, volume, data rights, and portability contractually clear?	Scalable agent terms, predictable economics, and exportable operational data.

A material gap does not automatically require replacement. It shows where the workflow may need an adapter, broker, human step, contractual change, or alternative system of action.

Build, buy, orchestrate, or wait

For each stack layer, there are four legitimate decisions:

- **Build** when the capability is differentiating or requires proprietary process logic.
- **Buy** when the capability is commodity and the vendor meets the operability requirements.
- **Orchestrate** when value comes from composing existing systems through a governed control plane.
- **Wait** when data, permissions, vendor capability, or economics make production delegation premature.

“Wait” is not failure. It is a decision to avoid funding a pilot whose constraints are already visible.

The principle to remember

> **A vendor is agent-ready only when its capabilities, permissions, evidence, and commercial model can be operated without borrowing a human and hiding the real cost.**

CHAPTER 9

The 90-day path to your first agent-operable workflow

90 DAYS · ONE WORKFLOW · ONE OPERATING MODEL

01 FRAME	02 MAP	03 ASSESS	04 DESIGN	05 VALIDATE	06 BUILD	07 OPERATE
1–10	11–25	26–40	41–55	56–65	66–80	81–90

The first workflow should teach the organization how to delegate safely. It should not attempt to prove the maximum capability of AI.

The following 90-day path is intentionally practical. It gives away most of the audit methodology because the value is not in knowing that the steps exist. The value is in running them with enough rigor and cross-functional accountability to produce a defensible decision.

Choose boring over impressive

A strong first workflow has:

- meaningful volume or latency;
- a clear current cost;
- repeatable inputs and outputs;
- an identifiable process owner;
- accessible evidence of quality;
- mostly reversible actions;
- limited external exposure;
- a bounded data domain;
- an available human escalation path;
- a sponsor willing to change the workflow, not only add a tool.

Good candidates include internal reconciliation, classification, compliance preparation, document intake, response drafting, quality assurance, knowledge routing, and workflow monitoring.

Weak first candidates include open-ended strategic decisions, public communications, high-value financial execution, employee discipline, legally binding commitments, and workflows whose basic rules are disputed.

Phase 1 — Frame the delegation case (Days 1–10)

Day 1–3: define the outcome

Write a one-sentence outcome in business terms.

Bad: “Use an AI agent for onboarding.” Better: “Reduce the median time from signed contract to implementation-ready account from five days to one, without increasing configuration or compliance errors.”

Then define:

- current cycle time and cost;

- target improvement;
- in-scope and out-of-scope cases;
- the user and stakeholders;
- the maximum acceptable consequence of error;
- the decision date for build, buy, or wait.

Day 4–6: appoint the pod

Name the sponsor, product owner, process owner, internal FDE or architect, engineering owner, identity/security owner, risk owner, and trust owner.

Do not proceed with “representatives will join when needed.” Agent operability fails in the gaps between functions. The accountable people need to be present when boundaries are designed.

Day 7–10: inventory the actions

List every verb the workflow performs. Avoid capability labels.

For each action record:

- object acted upon;
- current actor;
- system used;
- input and output;
- whether state changes;
- external exposure;
- approval or decision right;
- common exception;
- evidence left behind.

This action inventory becomes the spine of the assessment.

Phase 1 output: Delegation Case Brief + action inventory.

Phase 2 — Map the real workflow (Days 11–25)

Observe, do not only interview

Process documentation and stakeholder interviews are necessary but incomplete. Review actual cases, logs, artifacts, handoffs, and exceptions.

Ask experienced operators to narrate why they made decisions, not only what they clicked. Look for:

- steps performed outside the system;
- copied data and duplicate records;
- unofficial source-of-truth documents;
- workarounds used near deadlines;
- exception categories that are not written down;
- approvals obtained in chat or email;
- handoffs that depend on personal relationships;
- moments where the operator pauses because “something feels wrong.”

Draw three maps

1. **Documented map** — what the official process says.
2. **Observed map** — what actually happens.
3. **Agent candidate map** — where agents could read, draft, decide, or execute.

The difference between the first two is **process debt**. The uncertainty in the third is the operability work.

Identify the exception surface

The **exception surface** is the set of conditions under which the routine process stops being routine.

Capture:

- exception type;
- frequency;
- consequence;
- current resolver;
- evidence used;
- whether a policy exists;
- whether similar cases are decided consistently;
- required escalation time.

A workflow with a small, well-understood exception surface is a better first candidate than one where every case is “special.”

Phase 2 output: Verified current-state map + exception register.

Phase 3 — Run the three-layer assessment (Days 26–40)

Data shape assessment

For every action, map required context and its authority.

Classify sources across D0–D4. Identify the minimum shaping work needed for the target autonomy. Do not create a general data-cleanup backlog. Tie each issue to an action and consequence.

Process design assessment

Place every action into read, draft, decide, or execute. Score the five heat dimensions. Assign the default control posture. Identify de-escalators that reduce heat without destroying value.

Trust & permissions assessment

Draft the Agent Identity Card and permission envelope. Identify whether the agent acts as itself or on behalf of a user. Name sponsors, owners, escalation recipients, memory rules, and authorship requirements.

Find the operability ceiling

The target autonomy is capped by the weakest layer.

For each action, record:

- desired autonomy;
- current safe autonomy;
- limiting layer;
- required fix;
- owner and effort;
- evidence needed to promote autonomy later.

Phase 3 output: Agent Operability Map + gap register.

Phase 4 — Design the agent-ready workflow (Days 41–55)

Redraw the workflow as a human–agent operating model.

The target design should show:

- triggers and workflow states;
- agent, human, and system actors;
- data and context objects;
- actions and heat bands;
- approval and escalation points;
- tool calls and permission scopes;
- retry, dead-end, and rollback paths;
- authorship and audit artifacts;
- success, trust, and economic metrics.

Write the Trust Contract

Define:

- what the agent guarantees;
- what it may do automatically;
- what requires confirmation or named approval;
- what is forbidden;
- what uncertainty triggers escalation;
- what it remembers;
- what evidence it must leave;
- how it is paused and recovered;
- what incidents block operation.

Design for the minimum useful autonomy

Ask: what is the smallest form of delegation that creates a meaningful business result?

For an onboarding workflow, that might be:

- agent reads the contract and customer record;
- agent creates a complete onboarding pack;
- agent drafts system updates and missing-information requests;
- human approves customer-facing communication and non-standard configuration;
- agent executes approved internal updates;
- agent monitors completion and escalates delay.

This may deliver most of the cycle-time value without requiring autonomous external communication or financial commitment.

Phase 4 output: Agent-ready workflow + Trust Contract + Agent Identity Card.

Phase 5 — Validate the design before the build (Days 56–65)

Run the workflow against real historical cases and designed edge cases.

Test:

- routine success;
- missing and conflicting context;
- stale policy;
- unauthorized data request;
- high-value or bulk action;
- ambiguous user intent;
- unavailable system;
- duplicate trigger;
- prompt injection or untrusted instruction in retrieved content;
- incorrect but plausible recommendation;
- escalation recipient unavailable;

- partial failure after one state change;
- sponsor or approver change.

The objective is not to demonstrate that the model can handle everything. It is to verify that the operating model fails safely and visibly.

Produce a build, buy, or wait recommendation

Build if the workflow is differentiated, the stack is adequate, and the remaining gaps are tractable. **Buy** if a vendor meets the identity, permission, evidence, and commercial requirements. **Orchestrate** if the value lies in connecting existing tools under a shared trust layer. **Wait** if the critical layer would require disproportionate transformation or the economic case depends on unsafe autonomy.

Phase 5 output: Evidence-backed recommendation + scoped implementation path.

Phase 6 — Build the controlled slice (Days 66–80)

Build only the slice required to test the delegation case.

The implementation should include from day one:

- distinct identity;
- narrow tools and data scopes;
- structured context manifest;
- action limits and approval gates;
- runtime state and retries;
- authorship metadata;
- audit logging;
- pause and rollback controls;
- evaluation cases;
- operational dashboard for the owner.

Do not postpone these as “production hardening.” They are the product.

Establish golden cases

Create a representative set of cases with expected outcomes, acceptable variants, prohibited actions, and escalation expectations.

Golden cases should cover:

- common routine work;
- each exception category;
- each high-heat action;
- known failure modes;
- cases where the correct outcome is to ask, refuse, or escalate.

A system that only evaluates completed outputs will miss whether the agent should have stopped.

Phase 6 output: Controlled working slice + evaluation harness.

Phase 7 — Operate under supervision (Days 81–90)

Launch with a defined supervision model rather than an informal “human in the loop.”

Specify:

- which actions are reviewed every time;
- which are sampled and at what rate;
- who reviews and within what SLA;
- how corrections are recorded;
- how autonomy is paused;
- how incidents are classified;
- what evidence is required for promotion;
- what performance would trigger a wait or redesign decision.

CFO-proof success criteria

Measure the whole operating model.

Dimension	Example measure
Business	Cycle time, throughput, backlog, revenue or risk impact
Quality	Accuracy, completeness, rework, error severity
Trust	Approval, override, escalation, boundary violations
Operational	Runtime failures, retries, human review time
Economic	Net hours returned, cost per case, payback period
Adoption	Active use, abandonment, repeated delegation

A successful first workflow does not need to be autonomous. It needs to produce a better operating result with a control model the organization is willing to repeat.

The 90-day decision

At day 90, the organization should be able to answer:

- Which actions are agent-operable now?
- Which actions remain human and why?
- Which layer limits further autonomy?
- What evidence has the agent earned?
- What identity and permission pattern can be reused?
- What did the workflow actually save after supervision costs?
- Should we scale, redesign, buy, or wait?

The result is not merely one pilot. It is the first reusable piece of the organization’s delegation architecture.

The principle to remember

> **The first 90 days should prove an operating model, not a demo. Pick one workflow, cool the actions, make authority explicit, and leave behind reusable infrastructure.**

CONCLUSION

Operability compounds

The first agent-operable workflow is expensive because the organization is building more than an agent.

It is learning how to map reality rather than documentation. It is shaping context and authority. It is defining action heat. It is creating a distinct identity and permission envelope. It is deciding what authorship means when software contributes to work. It is building escalation, evidence, monitoring, and recovery. It is naming owners who previously sat in separate functions.

Done well, that work is not repeated from zero.

The second workflow can reuse:

- the agent identity pattern and sponsorship model;
- access review and expiry processes;
- the Action Heat Ladder and control postures;
- tool catalogue and approved connectors;
- context authority metadata;
- Trust Contract templates;
- authorship and audit schemas;
- trust gap taxonomy;
- evaluation harness and incident process;
- operating roles and pod model;
- vendor renewal criteria;
- the shared language of delegation.

This is why operability compounds.

Every captured failure becomes a permanent edge if it is turned into a named gap, policy rule, test case, de-escalator, or design pattern. Every successful delegation case gives the organization evidence for where autonomy can expand. Every workflow improves the paved road for the next one.

The durable advantage will not come from exclusive access to the smartest model. Capable models are increasingly available to everyone. The advantage will come from the organization's ability to turn its context, decisions, systems, and trust relationships into a governed environment that agents can operate.

The enterprise that wins is not the one with the most agents.

It is the one that can answer, for every consequential action:

- what the agent is doing;
- why it is allowed;
- what evidence it used;
- where the boundary sits;
- who can stop it;
- who owns the outcome;
- and what the organization learned from the last run.

> **Models make agents capable. Operability makes them useful. Trust makes them adoptable.**

APPENDIX A

The Agent Operability Self-Assessment

Use this checklist for one workflow, not for the enterprise as a whole.

Score each statement:

- **0 — Not true**
- **1 — Partly true / informal**
- **2 — Mostly true / documented**
- **3 — Consistently true / enforced and evidenced**

Layer 1 — Data shape

1. We can identify the authoritative source for every consequential decision.
2. Critical sources carry owner, status, scope, effective date, and review date.
3. The agent can distinguish policy, instruction, evidence, and historical example.
4. Important terms, states, and identifiers are consistent across systems.
5. Required data can be retrieved without manual browsing or copying.
6. Load-bearing tribal knowledge has been identified.
7. Decision residue is captured as governed examples rather than raw history.
8. Conflicting or missing authority triggers escalation.
9. Context is filtered to the task, user, client, market, and workflow instance.
10. Memory retention and cross-case boundaries are explicit.
11. Recommendations and actions can cite their evidence.
12. Freshness and supersession are machine-readable for critical sources.

Data shape subtotal: /36

Layer 2 — Process design

13. The workflow has been observed and verified, not only documented.
14. Triggers, success states, dead ends, retries, and exceptions are mapped.
15. Human, system, and agent actors are named with decision rights.
16. Inputs, intermediate artifacts, outputs, and audit artifacts are classified.
17. Every candidate agent action is written as a verb and object.
18. Every action is classified as read, draft, decide, or execute.
19. The five Action Heat dimensions have been scored.
20. The control posture matches the hottest dimension.
21. High-heat actions have been cooled through drafts, caps, sandboxes, allowlists, or undo.
22. Load-bearing judgment is distinguished from habitual manual work.
23. Escalation and human handoffs are designed with owners and service levels.
24. The target workflow defines the minimum useful autonomy.

Process design subtotal: /36

Layer 3 — Trust & permissions

25. The agent has a distinct, discoverable identity or governed delegated identity.
26. A human sponsor is accountable for lifecycle and access.
27. A product owner is accountable for behavior and outcomes.
28. A process owner is accountable for workflow truth and exceptions.
29. The permission envelope names resources, actions, conditions, limits, and duration.
30. The agent's own authority is separated from authority delegated by a user.
31. Prohibited actions are explicit and technically enforced where possible.
32. Memory policy defines scope, retention, editing, and deletion.
33. The authorship layer records agent, sources, policy, and human approvals.
34. Consequential actions can be reconstructed from the audit trail.
35. Access review, expiry, suspension, and retirement are defined.
36. Trust incidents trigger pause, investigation, correction, and policy improvement.

Trust & permissions subtotal: /36

Operating readiness

37. Quality, trust, operational, economic, and adoption metrics are defined.
38. Golden cases include routine work, exceptions, and "must escalate" outcomes.
39. Runtime monitoring detects boundary pressure, unusual access, and repeated failure.
40. The organization can pause the agent and restore the workflow safely.
41. The supervision model specifies what is approved, sampled, and reviewed.
42. Autonomy can be promoted or retracted based on evidence.

Operating readiness subtotal: /18

Scoring

0–35 — Watched, not worked The workflow may support search, summaries, or tightly supervised demonstrations. Production delegation would rely heavily on invisible human infrastructure.

36–70 — Draft-operable The workflow can support useful reading, classification, and drafting. Important context, exception, or permission gaps still limit decisions and execution.

71–90 — Bounded-operable The workflow can support recommendations and selected reversible actions inside a defined envelope. Remaining gaps should be tied to specific high-heat actions.

91–108 — Production-operable The workflow has strong foundations for governed execution, continuous monitoring, and evidence-based expansion of autonomy.

The ceiling rule

Do not rely only on the total score. Calculate each layer as a percentage. The lowest layer is the operability ceiling.

A workflow scoring 90 overall but only 45% in trust & permissions is not production-operable for high-heat action. The weakest layer determines the safe delegation posture.

APPENDIX B

Glossary: the vocabulary of agent operability

Action Heat

The consequence carried by an agent action if it goes wrong, assessed across reversibility, blast radius, exposure, commitment, and authority.

Action Heat Ladder

auxfirst's framework for mapping action heat to a control posture: auto-run, sampled review, propose + approve, named approver, or human execution.

Agent Identity Card

A one-page, human-readable definition of an agent's purpose, sponsor, owners, scope, data, tools, actions, permission envelope, trust stage, escalation, memory, authorship, and review lifecycle.

Agent operability

The capacity of a specific workflow to be performed by an agent with bounded autonomy, usable context, explicit decision rights, and reconstructable accountability.

Agent-operable enterprise

An organization that can repeatedly convert workflows into governed delegation cases using shared identity, permission, context, process, trust, and evidence infrastructure.

Answerability

The ability of the system and organization to explain and evidence what the agent attempted, was allowed to do, used as context, changed, escalated, and left behind — with named accountability.

Authorship layer

The metadata and evidence binding an agent-produced artifact or action to agent identity, initiating user or event, model and policy version, sources, approvals, final executor, and audit record.

Autonomy budget

The amount of independent action an agent has demonstrably earned for a workflow. Autonomy beyond evidence is assumed trust and increases the risk surface.

Context efficiency

The practice of giving an agent the minimum sufficient, authoritative, task-specific context rather than the maximum available context.

Context manifest

A record of the sources, versions, scopes, and exclusions supplied to an agent for a particular workflow run or decision.

Data shape

The degree to which information is findable, authoritative, structured, semantically clear, scoped, fresh, and linked to permissible action.

Decision residue

Governed traces of prior human decisions — situation, evidence, rationale, owner, scope, and validity — retained as context without treating historical precedent as universal policy.

Delegation architecture

The shared enterprise infrastructure and rules for assigning work and authority to non-human actors: identity, context, tools, permissions, policy, evidence, monitoring, and human accountability.

Delegation case

A defined business case specifying the workflow, actions delegated, required context, authority, controls, handoffs, evidence, accountability, and expected economic result.

De-escalator

A design move that cools an action: draft-only output, sandboxing, undo windows, hard caps, sampled review, or narrow allowlists.

Exception surface

The conditions under which a routine workflow becomes non-routine, including the frequency, consequence, resolver, evidence, and escalation needs of each exception.

Functional trust

Confidence that the agent can complete basic tasks reliably and predictably.

Contextual trust

Confidence that the agent understands relevant history, preferences, scope, and situational context.

Judgment trust

Confidence that the agent can make appropriate calls under ambiguity, including asking, refusing, and escalating.

Advocacy trust

Confidence that the agent will act in the user’s or organization’s intended interest when incentives, metrics, or objectives conflict.

Governed context

Context whose authority, freshness, scope, provenance, retention, and permissible use are continuously defined and enforced.

Headless readiness

The capacity of an enterprise application to expose useful states and actions through secure, documented, machine-operable interfaces with appropriate identity, permission, audit, and commercial support.

Human–agent operating model

A workflow design that assigns each action, decision, approval, escalation, and accountability to the appropriate human, agent, or deterministic system component.

Minimum useful autonomy

The lowest level of agent delegation that creates a meaningful business result. It is the preferred starting target for an operability design.

Operability ceiling

The maximum safe autonomy supported by the weakest of the three layers: data shape, process design, and trust & permissions.

Operability debt

The accumulated hidden work created when agents rely on ambiguous data, undocumented process logic, borrowed credentials, manual approvals, or incomplete evidence rather than durable operating infrastructure.

Permission envelope

The bounded combination of resources, actions, conditions, limits, autonomy posture, duration, delegation chain, escalation, and evidence requirements under which an agent may operate.

Process brain

A bounded, governed machine for one business process, built from a source of truth, named skills, policies, context, and outputs that can be run consistently rather than recreated through ad hoc prompting.

Progressive transparency

The AUX principle of showing more reasoning, evidence, and control while trust is being established or stakes are high, then tapering visible detail while preserving the underlying record.

Trust architecture

The design of identity, boundaries, evidence, control, escalation, authorship, monitoring, and recovery that makes agent action trustworthy and answerable.

Trust Contract

The agent’s operational constitution: scope, autonomy by action class, prohibitions, memory, evidence, escalation, guarantees, trust gaps, and incident posture.

Trust Harness

The runtime enforcement layer around an agent that applies deterministic policies, permission gates, escalation triggers, telemetry, and audit rules to probabilistic proposals and actions.

Trust incident

An event where agent behavior violates or materially pressures the intended contract, boundary, evidence requirement, or user relationship — even if no traditional security breach occurs.

Trust owner

The person accountable for autonomy boundaries, transparency, control, escalation, adoption, and the ongoing health of the human–agent relationship.

SELECTED REFERENCES

This field guide is an auxfirst framework informed by the following external standards, technical documentation, and enterprise observations.

1. **auxfirst — The AUX Manifesto and Agentic Experience Design.** Relationship design, trust stages, patterns, and principles for systems with memory, initiative, and judgment. https://auxfirst.com/
2. **auxfirst — The Action Heat Ladder.** Five heat dimensions, five control postures, and action cooling patterns. https://auxfirst.com/action-heat-ladder.html
3. **auxfirst — TrustKit & The Trust Harness.** Trust contracts, named trust gaps, autonomy fit, runtime enforcement, and continuous conformance. https://auxfirst.com/trustkit-harness.html
4. **auxfirst — Five Steps to Becoming an Agentic Organization.** Vision, ownership, process, technology, and the continuous AUX trust layer. https://auxfirst.com/5steps.html
5. **auxfirst — Agent Process Design.** The aggregate workforce view and the ten-stage design of an individual agent service. https://auxfirst.com/agentic-experience-center/agent-process-design.html
6. **Aaron Levie — notes from enterprise technology leaders on agent adoption.** Change management, operating-model modernization, and structured and unstructured data readiness. X, July 2026.
7. **NIST — Artificial Intelligence Risk Management Framework and Generative AI Profile.** Govern, map, measure, and manage AI risk across the lifecycle. https://www.nist.gov/itl/ai-risk-management-framework
8. **Microsoft Entra Agent ID — Agent identities and governance.** Distinct agent identities, sponsors, lifecycle, policy, and access governance. https://learn.microsoft.com/en-us/entra/agent-id/what-are-agent-identities
9. **Google Cloud IAM — Agent Identity.** First-class, verifiable agent identity and least-privilege access. https://docs.cloud.google.com/iam/docs/agent-identity-overview
10. **OWASP — Agentic Applications and AI Agent Security guidance.** Excessive agency, identity and privilege abuse, least privilege, input validation, human controls, isolation, and monitoring. https://genai.owasp.org/
11. **Model Context Protocol — architecture, authorization, tool security, and client best practices.** Secure boundaries, consent, access controls, validation, confirmation, rate limits, and audit logs. https://modelcontextprotocol.io/docs/
12. **Neil D. Lawrence — Data Readiness Levels.** A common language for the preparation and limitations of data used in machine learning projects. arXiv:1705.02245.

ABOUT AUXFIRST

auxfirst is the agency built for the agentic era. We help enterprise leaders, product teams, and developers design agents that remember context, adapt over time, and earn the right to act.

Our work connects Agentic Experience Design, process architecture, identity, permissions, trust contracts, and runtime controls — so AI moves from a capable demo into a governed operating model.

The next step: Agent Operability Audit

The **Agent Operability Audit** answers one practical question for one real workflow:

> **Is this workflow ready to be worked by an agent — or only watched by one?**

Over three weeks, auxfirst maps the workflow as it actually runs and audits the three layers:

- **Data shape** — what the agent can consume, what remains trapped in tribal knowledge, and what must be shaped before action.
- **Process design** — what the agent should read, draft, decide, and execute, using the Action Heat Ladder to set the right control posture.
- **Trust & permissions** — the agent identity, trust stage, permission envelope, authorship layer, escalation paths, and audit requirements.

You receive a current-versus-agent-ready workflow map, prioritized data fixes, a human–agent operating model, an Agent Identity Card, and a build / buy / orchestrate / wait recommendation with a scoped implementation path.

One workflow. Three weeks. Fixed scope.

This is not a data cleanup project, an AI strategy deck, or a pilot. It is the diagnostic that tells you whether the pilot will survive contact with your operating model before you fund it.

Start the conversation: https://auxfirst.com/

AGENT OPERABILITY — The Agent-Operable Enterprise First edition · July 2026
Designing trust into every layer.